REMARKS

Status:

The Examiner has objected to the use of the term "CAP file" in claims 1, 10 and 12 as an unclear reference to converted applet file.

Claims 7 and stand rejected under 35 U.S.C. §102(e) as being anticipated by the teaching of U. S. Patent No.6,324,685 to Balassanian. Claims 1, 10 and 12 stand rejected under 35 U.S.C. §103(a) as being unpatentable considering the teaching of Balassanian supplemented by the ordinary skill in the art. Claims 2-3, 8-9,11, 13-14 and 17-19 stand rejected under 35 U.S.C. §103(a) as being unpatentable considering the teaching of Balassanian in view of US Publication 2003/002868A1 to Schwabe et al.

Claims 1-19 are presented for reconsideration as is explained in the analysis that follows.

Analysis:

The specification is amended at the first paragraph of the "Field of the Invention" to further clarify the meaning of the term CAP file. This is not believed to add any new matter as "CAP file" has become a term of art, in its own right, through use by SUN Microsystems. See Appendix A hereof for comparable use in an excerpt from the article "An Introduction to JAVA Card Technology - Part 2, The JAVA Card Applet" downloaded from the SUN developers web site "developers.sun.com." It is believed this clarification overcomes the Examiner's objection.

Now considering the teaching of Balassanian, the order of events contemplated by the teaching appears to be detailed with respect to Balassanian Fig.3 and the discussion starting at col. 4, line 56. There is indeed a verification process at element 48. What is significantly different from Applicant's teaching, however, is that the original code taught by Balassanian is program code and the code that is verified is "*a common*

intermediate pseudo-binary version of the source applet code (intermediate form program module 44)". (See Balassanian col.5, lines 2-4)

Looking further to Balassanian, the discussion continues as follows:

"*The word pseudo is used because the intermediate form 44 is not processor specific* ***but is still a binary representation of the source program module 40.*** *This intermediate form can be re-targeted and compiled for a particular processor. Alternatively, the intermediate form 44 can be interpreted by an interpreter or virtual machine that understands the internal binary representation of the intermediate form. A target* ***compiler 52 compiles intermediate applet code 44 into an applet 54*** *in a processor specific format (binary) suitable for running natively on a given computing platform.*" (Italics added to indicate quotation, bolding added for emphasis)

Applicant, in contrast, reverse converts converted code back to a code file for verification.

Applicant addresses the code at the point where it is a converted applet(CAP file). Moreover, Applicant has recognized that, while the possible reconstruction of a code file may not result in an exact replicate, it is sufficiently consistent to support verification. Verification at a JAVA code level that permits use of commonly available verification tools.

What is missing from the Balassanian teaching (see again Fig. 3) is a process to go from box 54 to create the content of box 40 or 44. Applicant has recognized that when a CAP file is available, but its verification is uncertain, it is possible to satisfactorily back-up to a state that allows an effective verification. This aspect of the invention is clearly emphasized in all of the claims. The other prior art does not overcome the deficiency of Balassanian in this respect. Applicant's more detailed claims focus on more specific aspects of the reconstruction approach; such as,

preconversion using symbolic names and mapping with tokenized identifiers which support the back-up to a code file form. These claims provide protection at this more detailed level of processing the converted code.

Applicant has found a creative way to salvage CAP files, separate from their underlying JAVA code, which are not known to be verified. Moreover, it is not believed that the prior art taken alone or together teaches or suggests Applicants advance for verifying converted code files without resort to the actual program files from which they were created.

Consistant with the discussion above, Applicant solicits withdrawal of the rejections of claims and notice that this case has been placed in condition for allowance.

Respectfully Submitted,

George E. Grosser
Reg. No. 25,629l

c/o IBM Corp.
Dept. T81/Bldg. 503 PO Box 12195
Research Triangle Park, NC 27709
(919)968-7847 Fax 919-254-4330
EMAIL: gegch@prodigy.net

APPENDIX A

Article
An Introduction to Java Card Technology - Part 2. The Java Card Applet

by C. Enrique Ortiz
September 2003

Part 1 of this article covered the high-level aspects of Java Card technology - what smart cards are, the elements of a Java Card application, the communication aspects, and a summary of the different Java Card technology specifications. In this part we focus on the development aspects of Java Card applets: the typical steps when developing a Java Card application, the Sun Java Card Development kit, and the Java Card and Java Card RMI APIs.

Developing a Java Card Application

The typical steps when creating a Java Card application are:

1. *Write* the Java source.
2. *Compile* your source.
3. *Convert* the class files into a Converted Applet (CAP) file.
4. *Verify* that the CAP is valid; this step is optional.
5. *Install* the CAP file.

The first two steps are the same as when developing traditional programs in the Java programming language: write . . . files and compile them into . . . files. Once you have created Java Card class files, though, the process changes.

> The Java Card Virtual Machine (JCVM) is split into an off-card JVM and an on-card JVM. This split moves expensive operations off-card and allows for a small memory footprint on the card itself, but it results in extra steps when developing Java Card applications.

Before the Java Card classes can be loaded into a Java Card device, they must be converted to the standard CAP file format, and then optionally verified:

- Conversion entails transforming each Java package into a CAP file, which contains the combined binary representation of classes and interfaces in a package. Conversion is an off-card operation.
- Verification is an optional process to validate the CAP file for structure, valid bytecode subset, and inter-package dependencies. You may want to do verification on third-party vendor packages that you use, or if your converter tool comes from a third-party vendor. Verification is typically an off-card operation, but some card products may include an on-board verifier.

Once verified, the CAP file is ready to be installed on the Java Card device.

The Sun Java Card Development Kit

You can write Java Card applets, and even test them without a smart card or card reader, using the Sun Java Card Development Kit. This kit includes all the basic tools you need to develop and test Java Card applets:

- The Java Card Workstation Development Environment (JCWDE), a convenient, easy to use Java Card simulation tool that allows developers to execute class files directly, without having to convert and install CAP files. The JCWDE can be integrated with a debugger and IDEs.

> Starting with version 2.2.1 of the development kit, JCWDE supports Java Card RMI (JCRMI). Note that the JCWDE is not a full-blown Java Card simulator. It does not support a number of JCRE features, such as package installation, applet instance creation, firewalls, and transactions. Please refer to the development kit's User's Guide for more information.